# **UNIDAD DE TRABAJO 2: BASES DE DATOS RELACIONALES**

**TEMA 6: CONSULTAS (parte 2)**

**6.11. Introducción**

En este tema vamos a tratar el concepto de fila o “variables de tupla”. Al comenzar a utilizar múltiples tablas en la cláusula FROM de las sentencias SELECT, es una buena idea para empezar a utilizar las variables de tupla (también conocido como alias de tabla) de una manera coherente.



Veremos las “JOINS”, el “GROUP BY”, las funciones de grupo más usadas, el “HAVING” y terminaremos viendo tres operadores importantes en oracle: UNION, MINUS e INTERSECT.

**6.12. Variables de tupla**

Hasta ahora, nosotros formulábamos nuestras sentencias SQL de la siguiente forma:

select ename, init, job

from employees

where deptno = 20;

En realidad, esta afirmación es bastante incompleta. Ahora seremos un poco más precisos, debido a que los comandos SQL son cada vez un poco más complicados. Para ser completa y exacta deberíamos haber puesto algo así:

select e.ename, e.init, e.job

from employees e

where e.deptno = 20;

En este ejemplo, e es una variable tupla. “Tupla” es sólo un nombre para el término “fila” derivado de la teoría relacional. **En Oracle, las variables de tupla se conocen como alias de tabla.** Esa variable de tupla se declara en la cláusula FROM, inmediatamente después del nombre de la tabla, separada por espacios en blanco únicamente.

Una variable tupla siempre se extiende sobre una tabla o una expresión de tabla. En el ejemplo anterior “**e”** es una variable que representa una fila de la tabla empleados en cualquier momento. En el contexto de una fila específica, puede hacer referencia a los valores de una columna específica, como se muestra en la cláusula SELECT y WHERE del ejemplo. La variable tupla precede al nombre de la columna, separados por un punto.

**6.13. JOINS**

Se pueden especificar múltiples tablas en el FROM de una consulta. Vamos a probar qué ocurre con el siguiente ejemplo: nuestra intención es descubrir que empleados pertenecen a los departamentos:

select deptno, location, ename, init

from employees, departments;

¿Qué ocurre?, nos da un error:

ERROR at line 1:

ORA-00918: column ambiguously defined

197

El motor de Oracle no puede saber a qué columna “deptno” nos estamos refiriendo. Ambas tablas tienen una columna DEPTNO.

**Producto Cartesiano**

La siguiente select hace un segundo intento para encontrar los empleados que pertenecen a los departamentos. Si observamos el resultado, lo que obtenemos (al no introducir restricciones en la cláusula WHERE) es el producto cartesiano de ambas tablas: fijaros que nos salen 56 filas, 14 empleados y 4 departamentos.

select d.deptno, d.location, e.ename, e.init

from employees e, departments d;

**Combinaciones de Igualdad (equijoins)**

select d.deptno, d.location, e.ename, e.init

from employees e, departments d

where e.deptno = d.deptno

order by d.deptno, e.ename;

Esta es la select correcta, donde en el WHERE estamos especificando esa condición: una fila de una tabla y otra fila de la otra tabla siempre que la clave principal de una de las tablas coincida con la clave ajena en la otra tabla (relación).

**Convenciones de Diseño**

Las sentencias SQL deben ser correctas, por supuesto. Pero en el momento en que se hacen más largas y complicadas, se hace cada vez más importante adoptar un estilo de diseño determinado. Espacios en blanco adicionales (espacios, tabuladores y líneas nuevas) no tienen ningún significado en el lenguaje SQL, pero sin duda mejoran la legibilidad del código:

select d.deptno

, d.location

, e.ename

, e.init

from employees e

, departments d

where e.deptno = d.deptno

order by d.deptno

, e.ename;

Esta disposición ha demostrado ser muy útil en la práctica. Fíjate en la colocación de las comas al comienzo de la siguiente línea. Esto hace que agregar y quitar líneas sea más fácil. Cualquier otra norma está bien, también. Esto es sobre todo una cuestión de gusto. Sólo asegúrate de adoptar un estilo y utilizarlo de manera consistente.

**Combinaciones de No Igualdad**

Si utilizamos un operador de comparación que no sea un signo de igualdad en la cláusula WHERE, se llama una combinación de no igualdad. Para ver un ejemplo de una “no igualdad” veamos el siguiente ejemplo, que calcula el salario total anual para cada empleado.

select e.ename employee

, 12\*e.msal+s.bonus total\_salary

from employees e

, salgrades s

where e.msal between s.lowerlimit

and s.upperlimit;

**6.14. LA CLAÚSULA JOIN**

Los ejemplos de combinación anteriores utilizan el operador producto cartesiano (la coma en la cláusula FROM) como punto de partida y luego filtran las filas en la cláusula WHERE. No hay absolutamente nada de malo en este enfoque, y la sintaxis es totalmente compatible con la norma ANSI / ISO SQL estándar, pero la norma ANSI / ISO SQL estándar también es compatible con la sintaxis alternativa para especificar las joins.

En primer lugar, vamos a ver de nuevo la instrucción anterior:

select e.ename as employee

, m.ename as manager

from employees m

, employees e

where e.mgr = m.empno

and e.bdate > date '1965-01-01';

Se puede entender que la cláusula WHERE de la consulta contiene dos tipos de condiciones diferentes: la línea 5 contiene la condición de combinación para asegurarse de que se combinan las filas de la derecha, y la línea 6 es un "verdadero" (no-unión) para filtrar los empleados con base en las fechas de nacimiento.  
El siguiente listado muestra una consulta equivalente, produciendo los mismos resultados, utilizando una sintaxis de combinación diferente. Fijaros en las palabras clave JOIN y ON. También hay que tener en cuenta que esta sintaxis de combinación no utiliza comas en la cláusula FROM.

select e.ename as employee

, m.ename as manager

from employees m

JOIN

employees e

ON e.mgr = m.empno

where e.bdate > date '1965-01-01'

order by employee;

Esta sintaxis es más elegante que la sintaxis anterior, debido a que la unión está totalmente especificada en la cláusula FROM y la cláusula WHERE contiene sólo el filtrado.